```
#!/usr/bin/env python
#coding=utf-8

import sys, math, random, operator
from datetime import datetime
from numpy import *

class Stump:
  def __init__(self, err, threshold, s):
    self.err = err
    self.threshold = threshold
    self.s = s

  def __cmp__(self, other):
    return cmp(self.err, other.err)

# Decision stump: weak classifier
class DecisionStumpClassifier(object):
  def set_training_sample(self,X,Y):
    self.X, self.Y = X, Y

  def set_weights(self,w):
    self.weights = w

  def train(self):
    stumps = [self._build_stump_1d(x,self.Y,self.weights) for x in self.X.T]
    feature_index, best_stump = min(enumerate(stumps),key=operator.itemgetter(1))
    self.feature_index = feature_index
    self.stump = best_stump

  def predict(self,X,stump=None,feature_index=None):
    if stump is None:
      stump = self.stump
    if feature_index is None:
      feature_index = self.feature_index

    N,d = X.shape; s = stump.s; Y = ones(N)
    Y[where(X[:,feature_index]< stump.threshold)[0]] = -1
    Y[where(X[:,feature_index]>=stump.threshold)[0]] = 1
    return Y

  def _build_stump_1d(self,x,y,w):
    sorted_xyw = array(sorted(zip(x,y,w), key=operator.itemgetter(0)))
    xsorted = sorted_xyw[:,0]
    wy = sorted_xyw[:,1]*sorted_xyw[:,2]
    score_left, score_right = cumsum(wy), cumsum(wy[::-1])
    score = -score_left[0:-1:1] + score_right[-1:0:-1]
    Idec = where(xsorted[:-1]<xsorted[1:])[0]
    if len(Idec)>0:
      ind,maxscore = max(zip(Idec,abs(score[Idec])),key=operator.itemgetter(1))
      err = 0.5 - 0.5*maxscore
      threshold = (xsorted[ind]+xsorted[ind+1])/2
      s = sign(score[ind])
    else:
      err, threshold, s = 0.5, 0, 1
    return Stump(err, threshold, s)

class Adaboost(object):
  def set_training_sample(self,X,Y,w=None):
    self.X, self.Y = X, Y

  def train(self, T=10, threshold = 0.1):
    X = self.X
    Y = array(self.Y)
    N = len(self.Y)
    w = (1.0/N)*ones(N)
    self.alpha = []
    self.weak_classifier_settings = []
    weak_learner = DecisionStumpClassifier()
    weak_learner.set_training_sample(X,Y)
    for t in range(T):
      weak_learner.set_weights(w)
      weak_learner.train()
```

```
      Y_predict = weak_learner.predict(X) # predict h_t
      right_predict = abs(2-abs(Y-Y_predict))/2
      gamma_t = sum(w*right_predict) - 0.5     #calculate Advantage
      alpha_t = 0.5*log((1+2*gamma_t)/(1-2*gamma_t))
      w = exp(-alpha_t*Y*Y_predict); w /= sum(w)     #update D_t
      self.weak_classifier_settings.append({
           'stump':         weak_learner.stump,
           'feature_index': weak_learner.feature_index
           })
      self.alpha.append(alpha_t)
      error = self.calculate_error(self.predict(X,t), Y_predict)
      print "t:%d error:%f gamma:%f alpha:%f" % (t, error, gamma_t, alpha_t)
      self.T = t
      if error < threshold: # exit the algorithm if we get good enough results
        print "error %f < threshold %f, end" % (error, threshold); break

  def predict(self, X, t_len = None):
    X = array(X)
    N,d = X.shape
    Y = zeros(N)
    weak_learner = DecisionStumpClassifier()
    if t_len is None: t_len = self.T
    for t in range(t_len):
      settings = self.weak_classifier_settings[t]
      Y+=self.alpha[t]*weak_learner.predict(X, settings['stump'], settings['feature_
index'])
    Y[Y>0], Y[Y<0] = 1, -1
    return Y

  def calculate_error(self, Y, Y_pred):
    return sum(abs(Y-Y_pred)/2)/len(Y)

Ha = (
     1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
     3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
     4, 4, 5, 5, 5, 5, 5, 5, 5, 5,
     5, 6, 6, 6, 6, 7, 7, 7, 8, 8,
     9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
     9, 9,10,10,11,11,11,11,12,12,
    12,12,13,13,13,13,13,13,13,13,
    13,14,14,14,14,14,14,14,14,14,
    15,15,16,16,16,16,17,17,17,18,
    18,18,18,18,18,18,18,19,19,19,
    19,19,19,19,19,19,20,20,20,20,
    20,20,21,21,21,21,21,21,21
    )

Hb = (
     1, 2, 3, 4, 5, 6, 1, 2, 3, 4,
     1, 2, 3, 4, 5, 6, 7, 8, 9,10,
     1, 2, 1, 2, 3, 4, 5, 6, 7, 8,
     9, 1, 2, 3, 4, 1, 2, 3, 1, 2,
     1, 2, 3, 4, 5, 6, 7, 8, 9,10,
    11,12, 1, 2, 1, 2, 3, 4, 1, 2,
     3, 4, 1, 2, 3, 4, 5, 6, 7, 8,
     9, 1, 2, 3, 4, 5, 6, 7, 8, 9,
     1, 2, 1, 2, 3, 4, 1, 2, 3, 1,
     2, 3, 4, 5, 6, 7, 8, 1, 2, 3,
     4, 5, 6, 7, 8, 9, 1, 2, 3, 4,
     5, 6, 1, 2, 3, 4, 5, 6, 7
    )

Hv = (
       'b','c','x','f','k','s','f','g','y','s',
       'n','b','c','g','r','p','u','e','w','y',
       't','f','a','l','c','y','f','m','n','p',
       's','a','d','f','n','c','w','d','b','n',
       'k','n','b','h','g','r','o','p','u','e',
       'w','y','e','t','f','y','k','s','f','y',
       'k','s','n','b','c','g','o','p','e','w',
       'y','n','b','c','g','o','p','e','w','y',
       'p','u','n','o','w','y','n','o','t','c',
       'e','f','l','n','p','s','z','k','n','b',
```

```
        'h','r','o','u','w','y','a','c','n','s',
        'v','y','g','l','m','p','u','w','d'
    )

FIELDS = 21

def loadDataset():
  res = []
  line_num = 0
  file = open("data.txt")
  for line in file:
    raw = line.split(" ")
    if len(raw)<FIELDS: break
    line_num = line_num + 1
    res.append(1) if raw[0] == "p" else res.append(-1)
    for f in range(len(raw)):
      for i in xrange(0,len(Ha)):
        if(Ha[i]==(f+1) and Hv[i]==raw[f+1]):
          res.append(Hb[i])
          break
  res = array(res).reshape(line_num,FIELDS+1)
  return res

def get_training_set(data,size):
  return data[:size,1:],data[:size,0]

def get_test_set(data,size):
  return data[-size:,1:],data[-size:,0]

dataset = loadDataset()
for training_set_size in [500, 1000, 1500 ]:
  for test_set_size in [500, 1000, 1500]:
    Xtr,Ytr = get_training_set(dataset,training_set_size)
    Xte,Yte = get_test_set(dataset,test_set_size)
    ada = Adaboost()
    ada.set_training_sample(Xtr,Ytr)
    ada.train()
    Y_predict = ada.predict(Xte)
    print "size(%d,%d) AccuRate:%f" % \
        (training_set_size, test_set_size, 1-ada.calculate_error(Yte,Y_predict))
```